Type	Storage size
char	1 byte
int	4 bytes
short	2 bytes
long	8 bytes

Two's Complement Representation

1 1 0 1 = -8 + 4 + 1 = -3

$-2^3\ 2^2\ 2^1\ 2^0$

Range of n-bit 2's complement

$[-2^{(n-1)}, 2^{(n-1)} - 1]$

negative to positive:

invert all bits and "1"

Golden Rule of Alignment

针对对不能再细分的基本组成元素

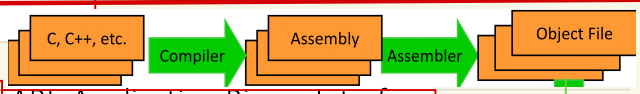
- Every (primitive) object starts at an address divisible by its size
- "Padding" is placed in between objects if needed

Structure Alignment

① 找到 Struct 中最大的 primitive

② 整个 Struct 需要从能整除该数的地址开始

③ 整个 Struct 占用的 bit 也要能被该数整除



ABI- Application Binary Interface

Library — linker — Executable

- X30 is the **link register** – used to hold return address
- X28 is **stack pointer** – holds address of top of stack
- X19-X27 are **callee-saved** – function must save these before writing to them
- X0-15 are **caller-saved** –function must save live values before call
- X0-X7 used for **arguments** (memory used if more space is needed)
- X0 used for **return value**

R type instructions (add '000', nor '001')

31-25	24-22	21-19	18-16	15-3	2-0
unused	opcode	regA	regB	unused	destR

J-type instructions (jalr '101')

Branch to — store PC+1

31-25	24-22	21-19	18-16	15-0
unused	opcode	regA	regB	unused

I type instructions (lw '010', sw '011', beq '100')

31-25	24-22	21-19	18-16	15-0
unused	opcode	regA	regB	offset

O type instructions (halt '110', noop '111')

31-25	24-22	21-0
unused	opcode	unused

lw: regB ← mem[regA value + offset]

sw: mem[regA value + offset] = regB value

如何提取中间一段 bits 例 15-10.

① X = X >> 10 ② X = X & 0x3F

	LC2K	LEG
# registers	8	32
Register width	32 bits	64 bits
Memory size	2^{18} bytes	2^{64} bytes
# instructions	8	40-ish
Addressability	Word	Byte

LC2K is **word addressable**

ARM is **byte addressable**

word is 4 byte

1. **Arithmetic**
   - Add, subtract, (multiply not in LEGv8)
2. **Data transfer**
   - Loads and stores—LDUR (load unscaled register), STUR, etc.
3. **Logical**
   - AND, ORR, EOR, etc.
   - Logical shifts, LSL, LSR
4. **Conditional branch**
   - CBZ, CBNZ, B.cond
5. **Unconditional branch (jumps)**
   - B, BR, BL

function calls

① Passing parameters

- Put the first few parameters in registers (if they fit) (X0 – X7)
- Put the rest in memory on the call stack— important concept

Call一个function Call Stack产生，return之后它 de-allocated

"caller-save" vs. "callee-save"

root function喜欢 callee save

leaf function喜欢 caller save

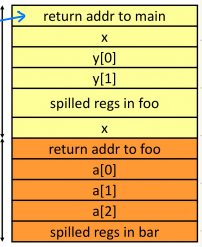
- Parameters
- Local variables
- Temporary storage
- Return address

```
void foo(){
  int a,b,c,d;

  a = 5; b = 6;
  c = a+1; d=c-1;
  save r1 to stack
  save r4 to stack
  bar();
  restore r1
  restore r4
  d = a+d;
  return();
}
```

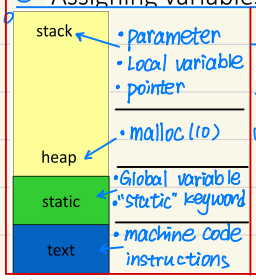
```
void foo(){
  int a,b,c,d;
  save r1
  save r2
  save r3
  save r4
  a = 5; b = 6;
  c = a+1; d=c-1;
  bar();
  d = a+d;
  restore r1
  restore r2
  restore r3
  restore r4
  return();
}
```

② Call Stack 图示

```
void foo()
{
  int x, y[2];
  bar(x);
}

void bar(int x)
{
  int a[3];
  printf();
}
```



③ Assigning variables to memory spaces



④ Stack pointer (SP):

register that keeps track of current top of stack

会指向当前方种的call stack的top. 并以此作为参考进行 address 计算.

```
sw   x30   [sp, #0]
sw   x1    [sp, #4]
sw   x2    [sp, #8]
```

R-instruction Encoding

Consider ADD X3, X4, X7

- R[Rd] = R[Rn] + R[Rm]
- Rd = X3, Rn = X4, Rm = X7

X3是Rd. 但在encode时出现在最后

opcode	Rm	shamt	Rn	Rd
11 bits	5 bits	6 bits	5 bits	5 bits

I-instruction Encoding

non-negative

ADDI X7, X5, #10

opcode	immediate	Rn	Rd
10 bits	12 bits	5 bits	5 bits

Unconditional Branching

branch	B 2500	go to PC + 10000	Branch to target address; PC-relative
branch to register	BR X30	go to X30	For switch, procedure return
branch with link	BL 2500	X30 = PC + 4; PC + 10000	For procedure call PC-relative

we don't specify X30 here → So X30 is implicitly been used

Call function

Logical Instructions

Category	InstructionExample		Meaning	Comments
Logical (gate)	and	AND X1, X2, X3	X1 = X2 & X3	Three reg. operands; bit-by-bit AND
	inclusive or	ORR X1, X2, X3	X1 = X2 | X3	Three reg. operands; bit-by-bit OR
	exclusive or	EOR X1, X2, X3	X1 = X2 ^ X3	Three reg. operands; bit-by-bit XOR
	and immediate	ANDI X1, X2, 20	X1 = X2 & 20	Bit-by-bit AND reg. with constant
	inclusive or immediate	ORRI X1, X2, 20	X1 = X2 | 20	Bit-by-bit OR reg. with constant
	exclusive or immediate	EORI X1, X2, 20	X1 = X2 ^ 20	Bit-by-bit XOR reg. with constant
shift	logical shift left	LSL X1, X2, 10	X1 = X2 << 10	Shift left by constant
	logical shift right	LSR X1, X2, 10	X1 = X2 >> 10	Shift right by constant

Branching far away

在和零相比时如何才能跳的更远.

```
CBZ   X15, FarLabel   (19 bits)
```

↓

```
CBNZ  X15, L1
B     FarLabel   (26 bits)
L1:   .....
```

Pseudo Instructions

- MOV X12, X2 ⟶ • ORR X12, XZR, X2

Memory Instructions

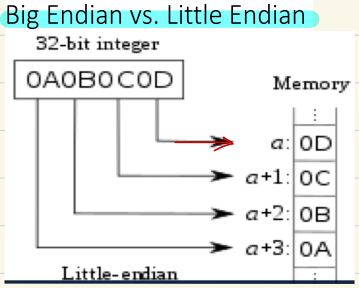
InstructionExample		Meaning	Comments
load register	LDUR X1, [X2,40]	X1 = Memory[X2 + 40]	Doubleword from memory to register
store register	STUR X1, [X2,40]	Memory[X2 + 40] = X1	Doubleword from register to memory
load signed word	LDURSW X1,[X2,40]	X1 = Memory[X2 + 40]	Word from memory to register
store word	STURW X1, [X2,40]	Memory[X2 + 40] = X1	Word from register to memory
load half	LDURH X1, [X2,40]	X1 = Memory[X2 + 40]	Halfword memory to register
store half	STURH X1, [X2,40]	Memory[X2 + 40] = X1	Halfword register to memory
load byte	LDURB X1, [X2,40]	X1 = Memory[X2 + 40]	Byte from memory to register
store byte	STURB X1, [X2,40]	Memory[X2 + 40] = X1	Byte from register to memory
move wide with zero	MOVZ X1,20, LSL 0	X1 = 20 or 20 * 2^{16} or 20 * 2^{32} or 20 * 2^{48}	Loads 16-bit constant, rest zeros
move wide with keep	MOVK X1,20, LSL 0	X1 = 20 or 20 * 2^{16} or 20 * 2^{32} or 20 * 2^{48}	Loads 16-bit constant, rest unchanged

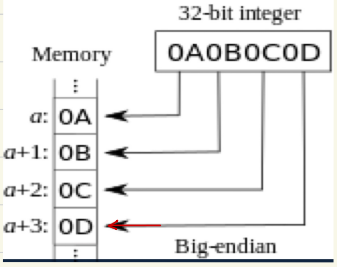
unscaled

load 从 R:register

store

Big Endian vs. Little Endian





Sequencing Instructions → Conditional Branch & unconditional Branch.

① One type compares a register to see if it is equal to zero.

compare and branch on equal 0	CBZ X1, 25	if (X1 == 0) go to PC + 100 (25×4)	Equal 0 test; PC-relative branch
compare and branch on not equal 0	CBNZ X1, 25	if (X1 != 0) go to PC + 100	Not equal 0 test; PC-relative branch
branch conditionally	B.cond 25	if (condition true) go to PC + 100	Test condition codes; if true, branch

② Another type checks the condition codes set in the status register.

FLAGS: NZVC record the results of (arithmetic) operations

Negative, Zero, oVerflow, Carry—not present in LC2K

Explicitly set ADD/SUB use "set"

例: ADDS X1, X2, X3 → X1=X2+X3 at the same time "set condition code"

Encoding	Name (& alias)	Meaning (integer)	Flags
0000 ==	EQ	Equal	Z==1
0001 !=	NE	Not equal	Z==0
1010 ≥	GE	Signed greater than or equal	N==V
1011 <	LT	Signed less than	N!=V
1100 >	GT	Signed greater than	Z==0 && N==V
1101 ≤	LE	Signed less than or equal	!(Z==0 && N==V)
1110	AL	Always	Any
1111	NV†		

How to use flag

```
CMP   X1, X2
B.GT  Label1
```

CMP is pseudo instruction = SUBS

- 这里的Instruction是 32 bits
- relative的address是 19 bits
- offset又是instruction的数量. 不用写地址的结果.

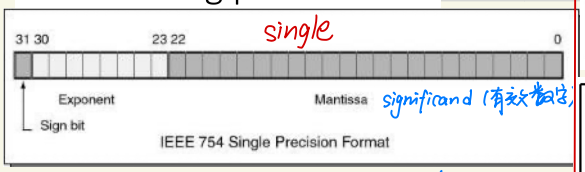
Object code format

- Header
- Text
- Data
- Symbol table
- Relocation table (maps symbols to instructions)

① Local 定义的 Global variable "D"
② Local 定义的 Function "T"
③ "extern" 在其它 file 中定义过的 GV "U"
④ "extern" 在其它 file 中定义的 Function "U"
⑤ "Static" variable "D"

① 调用方程: "Local" 和 "Global" 都算
② Reference variable 的值
其中 local 变量永远不出现 & 不管 declare.
状态有 "LDUR" "SDUR" "BL"

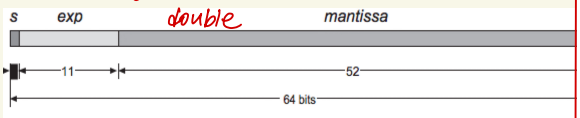
IEEE Floating point format

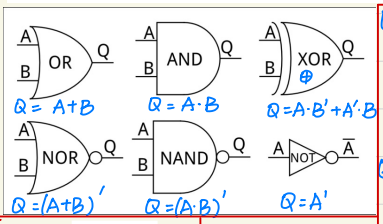


- Add 127 to the value of the exponent to encode: 0.625 ×2
- -127 → 00000000 1 → 10000000
- -126 → 00000001 2 → 10000001
- ... ...
- 0 → 01111111 128 → 11111111

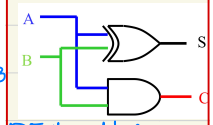
0.25 ×2 ··· 1
0.5 ×2 ··· 0
0.0 ··· 1

$10.625_{10} \rightarrow 10 \rightarrow 1010$
$0.625 \rightarrow .101$
1010.101_2
$1010.101_2 = 1.01010_2 \times 10^3$
Exponent = 3
ignore since must be 1 Mantissa = 01010

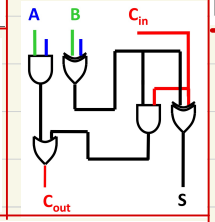


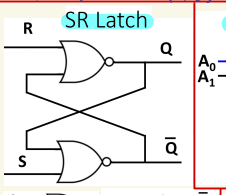


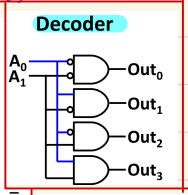
① Sub-adder

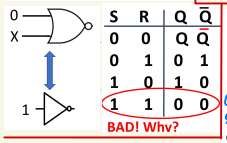


② Full-adder



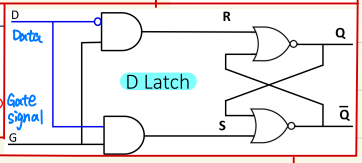




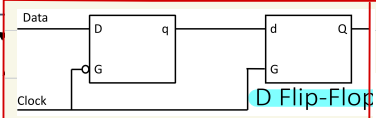


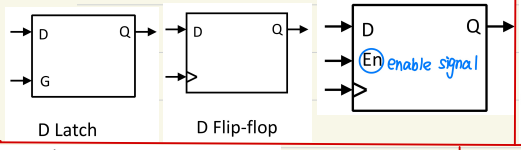
S	R	Q	Q̄
0	0	Q	Q̄
0	1	0	1
1	0	1	0
1	1	0	0

BAD! Why?

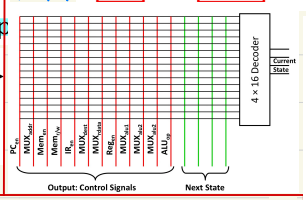


D	G	Q	Q̄
0	0	Q	Q̄
0	1	0	1
1	0	Q	Q̄
1	1	1	0

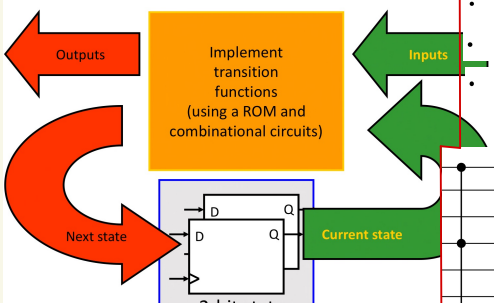




Control ROM



Implementing an FSM

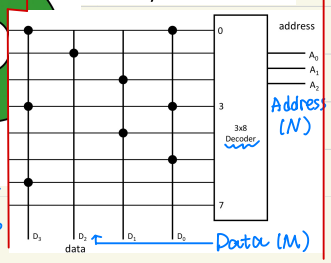


size of ROM: $2^N \cdot M$ | table 相结合的方式
通常使用 combinational logic 和 lookup

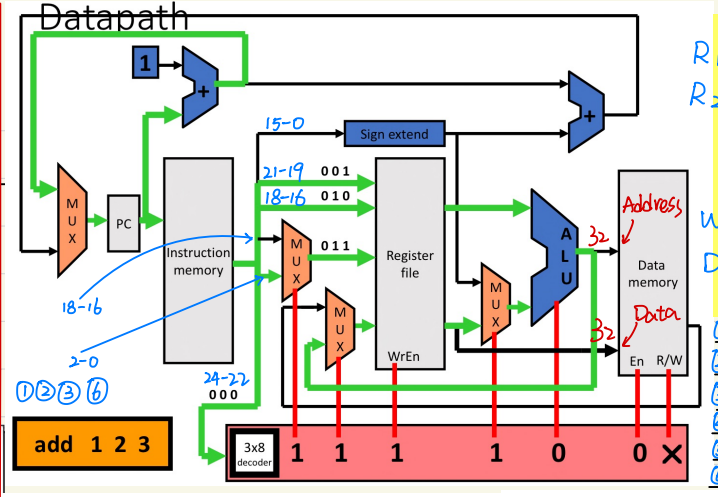
Building the controller

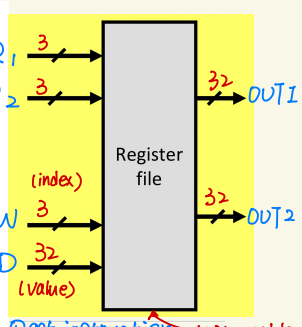
- Finite State Machine
- Read-Only Memory
- Custom combinational circuits

8-entry 4-bit ROM

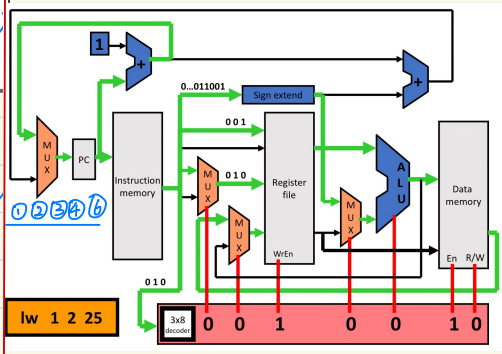


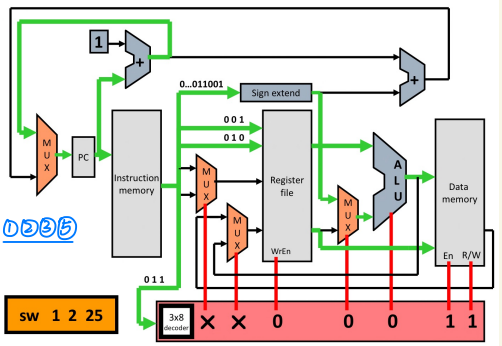
Datapath

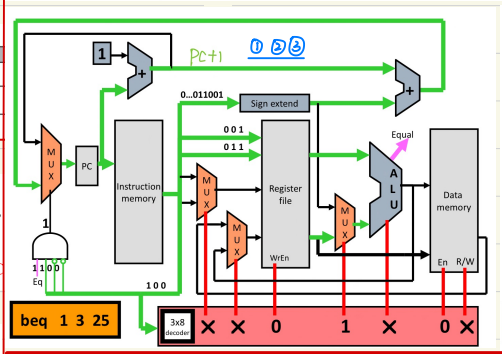


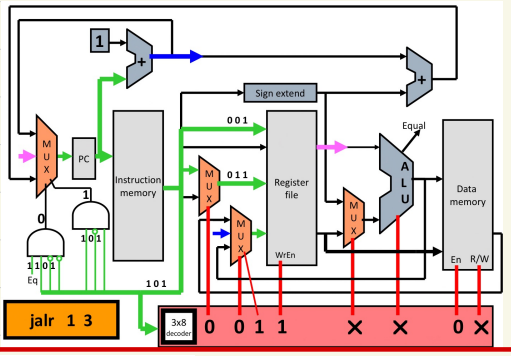


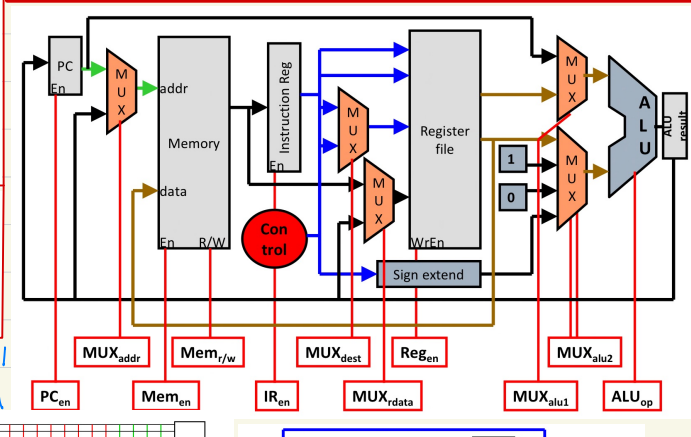
① get instruction
② read register
③ ALU
④ Read from memory
⑤ Load to memory
⑥ write to register

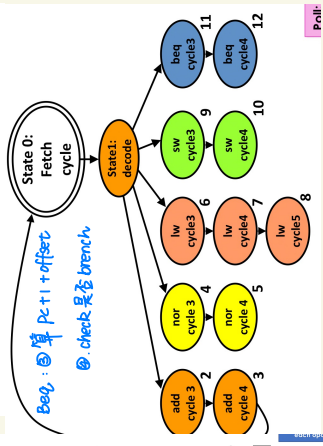


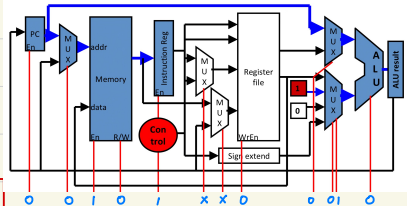


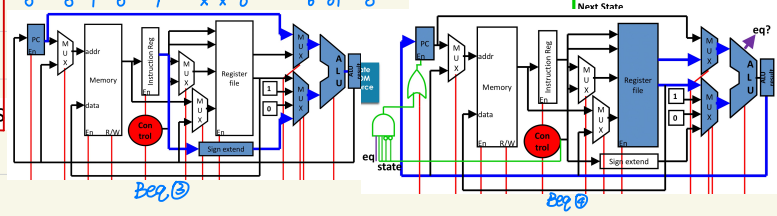












0000 000 | 24-22 | 21-19 | 18-16
opcode | B | B | A
15-0 offset